UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/783,343	02/20/2004	David Wortendyke	MS1-1825US	7693

22801 7590 03/06/2009

LEE & HAYES, PLLC
601 W. RIVERSIDE AVENUE
SUITE 1400
SPOKANE, WA 99201

EXAMINER	
SYED, FARHAN M	

ART UNIT	PAPER NUMBER
2165	

MAIL DATE	DELIVERY MODE
03/06/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/783,343	WORTENDYKE ET AL.	
	Examiner	Art Unit	
	FARHAN M. SYED	2165	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *19 December 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-38* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-38* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *(See Office Action)*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims1-38 are pending. The Examiner acknowledges amended claims 1, 7, 15, 24, and 35.

Continued Examination Under 37 CFR 1.114

2. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on 19 December 2008 has been entered.

Information Disclosure Statement

3. The information disclosure statement (IDS) submitted on 19 December 2008 is being considered by the examiner.

Response to Remarks/Argument

4. Applicant's arguments see pages 14-15, filed 14 April 2008, with respect to claims 7-23 have been fully considered and are persuasive. The 35 U.S.C. 101 rejection of a non-final office action, mailed 13 December 2007, has been withdrawn.

5. Applicant's arguments with respect to claims 1-38 have been considered but are moot in view of the new ground(s) of rejection.

The Examiner's rejections of the claims, now set forth are in light of the applicant's arguments against the art applied, But applied in the modified position therefore, the arguments are deemed moot.

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. Claims 1-38 are rejected under 35 U.S.C. 103(a) as being anticipated by Campailla, Alexis (U.S. 7,136,899; filed 11 December 2000, and known hereinafter as Campailla)(newly presented, see IDS submission on 26 September 2007) in view of a non-patent literature titled "Efficient Filtering of XML Documents for Selective Dissemination of Information" by Mehmet Altinel, et al., 26th VLDB Conference, 2000, pages 53-64 (previously presented and known hereinafter as Altinel), and in further view of a non-patent literature titled "On Efficient Matching of Streaming XML Documents and Queries" by Sailaja et al, University of British Columbia, Canada, 2002, pages 1-20 (previously presented and known hereinafter as Sailaja).

As per claim 1, Campailla teaches a method comprising: receiving an input (i.e. input message queue)(See Figure 3; column 5, lines 50-64), wherein the input comprises a plurality of characters (i.e. "*The input message queue model receives the sequence of information messages from a publisher message system…the sequence of information messages include stock quote data, such as a stock quote symbol, etc….*")(See Figure 3; column 5, lines 50-64); grouping the plurality of characters into one or more elemental language units (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."* The preceding text clearly indicates that the logical query evaluation process includes the step of grouping the plurality of characters into one or more elemental language units.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); breaking the one or more elemental language units into one or more constituent parts (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."*The preceding text clearly indicates that the logical query evaluation process indicates breaking one or more elemental language units into one or more constituent parts.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); generating opcodes from the one or more elemental language units and from the one or more constituent parts (i.e. *"Each of the inverse query subscription modules applies the subscriber's information request criteria to the information..."*)(see Figures 3 and 4; column 6, lines 13-16), wherein the language units have been parsed and compiled into opcodes (Figure 10 illustrates the parsed language units and compiled into opcode nodes.)(See Figures 3, 4, 8, and 10); merging the opcodes into an opcode tree comprising opcode nodes and branch nodes, wherein there are no opcodes added to the opcode tree during an active merging (i.e. Figure 5 appears to create an opcode tree, which is further illustrated in Figure 6, where opcode nodes and branches are illustrated as P1, P2,

P3)(See at least Figures 3, 5, 6, 9) traversing an opcode tree (i.e. inverse query decision that is contained within a binary decision diagram)(see Figures 1, 2, and 3; see also column 2, lines 33-45) that includes a plurality of opcode nodes which together define opcodes that should be executed to evaluate a plurality of queries (i.e. *"binary decision diagrams" "evaluates a first information request criteria based upon information within the received messages, evaluates one or more information request criteria based upon information within the received messages using the identified logical implications between one or more binary decision diagrams"* The preceding text clearly indicates that evaluating the information request criteria is to traverse an opcode three that includes a plurality of opcodes.)(column 2, lines 33-45); and wherein a tree segment in a shared path represents an opcode block prefix that is common to two or more queries (i.e. Figure 2 appears to illustrate a tree segment, i.e. P1, P2 in a shared path represents an opcode block prefix, i.e. <rop>, that is common to two or more queries)(Figures 2, 3, 6, and 7).

Campailla does not explicitly teach executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled.

Altinel teaches executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input (i.e. *"The Query Index is used to match documents to individual XPath **queries**." "The events that drive the execution of the Filter Engine are generated by the XML Parser (as described in the following section). In the XFilter **execution** model, a profile is considered to match a document when the final state of its FSM is reached. The Query Index is built over the states of the XPath queries."* The preceding text clearly indicate that execution takes place within the Xfilter execution model, where opcode

nodes, which are nodes within an XML documents, and plurality of queries are Xpath queries.)(Page 56, section 4.1); and wherein a relationship between the opcodes (nodes in an XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) and the opcode tree (i.e. XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) is embedded in the opcodes that is created when a query is compiled (see XFilter engine which uses a sophisticated index structure and a modified Finite State Machine approach and represent queries using XPath language)(page 53).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Campailla with the teachings of Altinel to include executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

The combination of Campailla and Altinel do not explicitly teach the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing.

Sailaja teaches the method wherein the input comprises a plurality of characters (i.e. Figure 1 illustrates the elemental language units, where Parts (Q), (P,R), are elemental language units.)(Figure 1; Sections 1, 3, and 4); evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time (i.e. *"A more clever approach is to devise algorithm that makes a* ***constant number of passes*** *over the document and determine the queries answered by each of its element…"* The Examiner the elements are common query expressions, evaluating the input against multiple queries as determine the queries. The Examiner further understands the limitation of parallel as a constant number of passes, because a naïve way to process queries one at a time is inefficient.)(Section 1, see Example 11); generating at least some of the elemental language units into opcodes (i.e. Figure 1 further exemplifies the generating at least some element units into opcode, where Part (Q), which exemplifies elemental language unit and opcodes which is exemplified by 'Name', 'Brand'. Since Sailaja focuses on XML, an ordinary person skilled in the art understands that XML tags are sets of instructions that are used to create 'Name' and 'Brand,' but not necessarily limited to just creating 'Name' and 'Brand'.)(Figure 1, Sections 1 and 4); hierarchical nature (i.e. Figure 1 clearly illustrates the hierarchical nature, which is a data tree node showing query labeling. To avoid cluttering of Figure 1, node numbers were omitted, however further explained in Example 11.)(Figure 1; Section 1); maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text and Figures 3 and 4 illiustrates maintaining the opcode tree, which is tracking queries that have been answered and making a copy of the opcode tree is the three different lists QL, CML, and PL which are copies of the data tree node.)(Figure 3; Section 4);

and updating the opcode tree copy (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text clearly indicates the use of the auxillary list, PL, which pushes the list. An ordinary person skilled in the art understands that when a push is made, it clearly states that an update is being performed.)(Figure 3; Section 4).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the combined teachings of Campailla and Altinel with the teachings of Sailaja to include the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing; and updating the opcode tree copy with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

As per claim 2, both Campailla and Altinel teach a method, the executing step further comprising using an intermediate result to execute an opcode node when the opcode node includes one or more ancestor opcode nodes that have been executed to derive the intermediate result (Page 56, section 4).

As per claim 3 both Campailla and Altinel teach a method, the executing step further comprising executing a single opcode node to evaluate at least a portion of at least two of the plurality of queries (Page 54, section 2.1, 2.2; page 56, section 4, 4.1).

As per claim 4, both Campailla and Altinel teach, the multiple queries further comprising Xpath queries (i.e. *"The Query Index is used to match documents to individual Xpath queries.")*(Page 56, section 4, 4.1).

As per claim 5, both Campailla and Altinel teach, the executing step further comprising executing a branch node to execute multiple opcode nodes that depend from the branch node, the branch node including an indexed branch lookup function (i.e. Figure 5 depicts an example of executing multiple nodes in Figure 5, section a, titled Example Queries and Corresponding Path Nodes, where the path nodes are multiple opcode nodes that depend from the branch node.)(Page 59, Figure 5).

As per claim 6, both Campailla and Altinel teach, further comprising a hash function as the indexed branch lookup procedure (i.e. Figure 5 depicts an example of a hash table. An ordinary person skilled in the art anticipates the use of a hash function when using a hash table.)(Page 59, Figure 5).

As per claims 7, Campailla teaches an opcode tree data structure stored on one or more computer-readable media having computer executable instructions stored on a

computing device (See Figure 2), comprising: receiving an input (i.e. input message queue)(See Figure 3; column 5, lines 50-64), wherein the input comprises a plurality of characters (i.e. "*The input message queue model receives the sequence of information messages from a publisher message system…the sequence of information messages include stock quote data, such as a stock quote symbol, etc….*")(See Figure 3; column 5, lines 50-64); grouping the plurality of characters into one or more elemental language units (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."* The preceding text clearly indicates that the logical query evaluation process includes the step of grouping the plurality of characters into one or more elemental language units.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); breaking the one or more elemental language units into one or more constituent parts (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."*The preceding text clearly indicates that the logical query evaluation process indicates breaking one or more elemental language units into one or more constituent parts.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); generating opcodes from the one or more elemental language units and from the one or more constituent parts (i.e. *"Each of the inverse query subscription modules applies the subscriber's information request criteria to the information..."*)(see Figures 3 and 4; column 6, lines 13-16), wherein the language units have been parsed and compiled into opcodes (Figure 10 illustrates the parsed language units and compiled into opcode nodes.)(See Figures 3, 4, 8, and 10); merging the opcodes into an opcode tree comprising opcode nodes and branch nodes, wherein there are no opcodes added to the opcode tree during an active merging (i.e. Figure 5 appears to create an opcode tree, which is further illustrated in Figure 6, where opcode nodes and branches are illustrated as P1, P2,

P3)(See at least Figures 3, 5, 6, 9) traversing an opcode tree (i.e. inverse query decision that is contained within a binary decision diagram)(see Figures 1, 2, and 3; see also column 2, lines 33-45) that includes a plurality of opcode nodes which together define opcodes that should be executed to evaluate a plurality of queries (i.e. *"binary decision diagrams" "evaluates a first information request criteria based upon information within the received messages, evaluates one or more information request criteria based upon information within the received messages using the identified logical implications between one or more binary decision diagrams"* The preceding text clearly indicates that evaluating the information request criteria is to traverse an opcode three that includes a plurality of opcodes.)(column 2, lines 33-45); and wherein a tree segment in a shared path represents an opcode block prefix that is common to two or more queries (i.e. Figure 2 appears to illustrate a tree segment, i.e. P1, P2 in a shared path represents an opcode block prefix, i.e. <rop>, that is common to two or more queries)(Figures 2, 3, 6, and 7).

Campailla does not explicitly teach executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled.

Altinel teaches executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input (i.e. *"The Query Index is used to match documents to individual XPath **queries**." "The events that drive the execution of the Filter Engine are generated by the XML Parser (as described in the following section). In the XFilter **execution** model, a profile is considered to match a document when the final state of its FSM is reached. The Query Index is built over the states of the XPath queries."* The preceding text clearly indicate that execution takes place within the Xfilter execution model, where opcode

nodes, which are nodes within an XML documents, and plurality of queries are Xpath queries.)(Page 56, section 4.1); and wherein a relationship between the opcodes (nodes in an XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) and the opcode tree (i.e. XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) is embedded in the opcodes that is created when a query is compiled (see XFilter engine which uses a sophisticated index structure and a modified Finite State Machine approach and represent queries using XPath language)(page 53).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Campailla with the teachings of Altinel to include executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

The combination of Campailla and Altinel do not explicitly teach the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing.

Sailaja teaches the method wherein the input comprises a plurality of characters (i.e. Figure 1 illustrates the elemental language units, where Parts (Q), (P,R), are elemental language units.)(Figure 1; Sections 1, 3, and 4); evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time (i.e. *"A more clever approach is to devise algorithm that makes a* ***constant number of passes*** *over the document and determine the queries answered by each of its element…"* The Examiner the elements are common query expressions, evaluating the input against multiple queries as determine the queries. The Examiner further understands the limitation of parallel as a constant number of passes, because a naïve way to process queries one at a time is inefficient.)(Section 1, see Example 11); generating at least some of the elemental language units into opcodes (i.e. Figure 1 further exemplifies the generating at least some element units into opcode, where Part (Q), which exemplifies elemental language unit and opcodes which is exemplified by 'Name', 'Brand'. Since Sailaja focuses on XML, an ordinary person skilled in the art understands that XML tags are sets of instructions that are used to create 'Name' and 'Brand,' but not necessarily limited to just creating 'Name' and 'Brand'.)(Figure 1, Sections 1 and 4); hierarchical nature (i.e. Figure 1 clearly illustrates the hierarchical nature, which is a data tree node showing query labeling. To avoid cluttering of Figure 1, node numbers were omitted, however further explained in Example 11.)(Figure 1; Section 1); maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text and Figures 3 and 4 illiustrates maintaining the opcode tree, which is tracking queries that have been answered and making a copy of the opcode tree is the three different lists QL, CML, and PL which are copies of the data tree node.)(Figure 3; Section 4);

and updating the opcode tree copy (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text clearly indicates the use of the auxillary list, PL, which pushes the list. An ordinary person skilled in the art understands that when a push is made, it clearly states that an update is being performed.)(Figure 3; Section 4).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the combined teachings of Campailla and Altinel with the teachings of Sailaja to include the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing; and updating the opcode tree copy with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

As per claim 8, both Campailla and Altinel teach, the executing step further comprising executing a branch node to execute multiple opcode nodes that depend from the branch node, the branch node including an indexed branch lookup function (i.e. Figure 5 depicts an example of executing multiple nodes in Figure 5, section a, titled Example Queries

and Corresponding Path Nodes, where the path nodes are multiple opcode nodes that depend from the branch node.)(Page 59, Figure 5).

As per claim 9, both Campailla and Altinel teach, further comprising a hash function as the indexed branch lookup procedure (i.e. Figure 5 depicts an example of a hash table. An ordinary person skilled in the art anticipates the use of a hash function when using a hash table.)(Page 59, Figure 5).

As per claim 10, both Campailla and Altinel teach, the multiple queries further comprising Xpath queries (i.e. *"The Query Index is used to match documents to individual Xpath queries.")*(Page 56, section 4, 4.1).

As per claim 11, Altinel teaches an opcode tree data structure, further comprising at least one shared segment that corresponds to multiple queries (Page 56, section 4, 4.1; page 57, section 4.2).

As per claim 12, both Campailla and Altinel teach an opcode tree data structure, wherein a single execution of the shared segment evaluates at least a portion of each of the multiple queries (Page 56, section 4.1).

As per claim 13, both Campailla and Altinel teach an inverse query engine containing the opcode tree data structure (i.e. *"For this purpose, similar to traditional SDI*

systems, the Filter Engine component of Xfilter contains an inverted index, called the Query Index.")(Page 56, section 4.1).

As per claim 14, both Campailla and Altinel teach an opcode tree data structure, further comprising a branch node that includes references to more than two dependent opcode nodes (i.e. Figure 3a, titled Example Queries and Corresponding Path Nodes clearly indicate that a branch node is '//b' and more than two dependent opcode nodes are /*/c/d)(Page 57, Figure 3).

As per claim 15, Campailla teaches a query evaluation system comprising: receiving an input (i.e. input message queue)(See Figure 3; column 5, lines 50-64), wherein the input comprises a plurality of characters (i.e. "*The input message queue model receives the sequence of information messages from a publisher message system…the sequence of information messages include stock quote data, such as a stock quote symbol, etc….*")(See Figure 3; column 5, lines 50-64); grouping the plurality of characters into one or more elemental language units (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."* The preceding text clearly indicates that the logical query evaluation process includes the step of grouping the plurality of characters into one or more elemental language units.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); breaking the one or more elemental language units into one or more constituent parts (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."*The preceding text clearly indicates that the logical query evaluation process indicates breaking one or more elemental language units into one or more constituent parts.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-

55); generating opcodes from the one or more elemental language units and from the one or more constituent parts (i.e. *"Each of the inverse query subscription modules applies the subscriber's information request criteria to the information..."*)(see Figures 3 and 4; column 6, lines 13-16), wherein the language units have been parsed and compiled into opcodes (Figure 10 illustrates the parsed language units and compiled into opcode nodes.)(See Figures 3, 4, 8, and 10); merging the opcodes into an opcode tree comprising opcode nodes and branch nodes, wherein there are no opcodes added to the opcode tree during an active merging (i.e. Figure 5 appears to create an opcode tree, which is further illustrated in Figure 6, where opcode nodes and branches are illustrated as P1, P2, P3)(See at least Figures 3, 5, 6, 9) traversing an opcode tree (i.e. inverse query decision that is contained within a binary decision diagram)(see Figures 1, 2, and 3; see also column 2, lines 33-45) that includes a plurality of opcode nodes which together define opcodes that should be executed to evaluate a plurality of queries (i.e. *"binary decision diagrams" "evaluates a first information request criteria based upon information within the received messages, evaluates one or more information request criteria based upon information within the received messages using the identified logical implications between one or more binary decision diagrams"* The preceding text clearly indicates that evaluating the information request criteria is to traverse an opcode three that includes a plurality of opcodes.)(column 2, lines 33-45); and wherein a tree segment in a shared path represents an opcode block prefix that is common to two or more queries (i.e. Figure 2 appears to illustrate a tree segment, i.e. P1, P2 in a shared path represents an opcode block prefix, i.e. <rop>, that is common to two or more queries)(Figures 2, 3, 6, and 7).

Campailla does not explicitly teach executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes

and the opcode tree is embedded in the opcodes that is created when a query is compiled.

Altinel teaches executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input (i.e. *"The Query Index is used to match documents to individual XPath **queries**." "The events that drive the execution of the Filter Engine are generated by the XML Parser (as described in the following section). In the XFilter **execution** model, a profile is considered to match a document when the final state of its FSM is reached. The Query Index is built over the states of the XPath queries."* The preceding text clearly indicate that execution takes place within the Xfilter execution model, where opcode nodes, which are nodes within an XML documents, and plurality of queries are Xpath queries.)(Page 56, section 4.1); and wherein a relationship between the opcodes (nodes in an XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) and the opcode tree (i.e. XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) is embedded in the opcodes that is created when a query is compiled (see XFilter engine which uses a sophisticated index structure and a modified Finite State Machine approach and represent queries using XPath language)(page 53).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Campailla with the teachings of Altinel to include executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled with the motivation to perform

efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

The combination of Campailla and Altinel do not explicitly teach the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing.

Sailaja teaches the method wherein the input comprises a plurality of characters (i.e. Figure 1 illustrates the elemental language units, where Parts (Q), (P,R), are elemental language units.)(Figure 1; Sections 1, 3, and 4); evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time (i.e. *"A more clever approach is to devise algorithm that makes a* ***constant number of passes*** *over the document and determine the queries answered by each of its element…"* The Examiner the elements are common query expressions, evaluating the input against multiple queries as determine the queries. The Examiner further understands the limitation of parallel as a constant number of passes, because a naïve way to process queries one at a time is inefficient.)(Section 1, see Example 11); generating at least some of the elemental language units into opcodes (i.e. Figure 1 further exemplifies the generating at least some element units into opcode, where Part (Q), which exemplifies elemental language unit and opcodes which is exemplified by 'Name', 'Brand'. Since Sailaja focuses on XML, an ordinary person skilled in the art understands that XML tags are sets of instructions that are used to create 'Name' and 'Brand,' but not necessarily limited to just creating 'Name' and 'Brand'.)(Figure 1, Sections 1 and 4); hierarchical nature (i.e. Figure 1 clearly illustrates the hierarchical nature, which is

a data tree node showing query labeling. To avoid cluttering of Figure 1, node numbers were omitted, however further explained in Example 11.)(Figure 1; Section 1); maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text and Figures 3 and 4 illiustrates maintaining the opcode tree, which is tracking queries that have been answered and making a copy of the opcode tree is the three different lists QL, CML, and PL which are copies of the data tree node.)(Figure 3; Section 4); and updating the opcode tree copy (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text clearly indicates the use of the auxillary list, PL, which pushes the list. An ordinary person skilled in the art understands that when a push is made, it clearly states that an update is being performed.)(Figure 3; Section 4).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the combined teachings of Campailla and Altinel with the teachings of Sailaja to include the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree,

wherein operations may be undertaken on the opcode tree without interfering with a query processing; and updating the opcode tree copy with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

As per claim 16, both Campailla and Altinel teach a query evaluation system, further comprising an input module configured to receive an input (i.e. *"There are two main sets of inputs to the system: user profiles and data items (i.e., documents). User profiles describe the information preferences of individual users. In most systems these profiles are created by the users, typically by clicking on items in a Graphical User Interface."* The preceding text clearly indicates that receiving an input is an input to the system by the user.)(Page 54, section 2.1) that is evaluated against each of the plurality of queries when the query processor executes the opcode nodes (i.e. *"The Query Index is used to match documents to individual XPath **queries**."* *"The events that drive the execution of the Filter Engine are generated by the XML Parser (as described in the following section). In the XFilter **execution** model, a profile is considered to match a document when the final state of its FSM is reached. The Query Index is built over the states of the XPath queries."* The preceding text clearly indicate that execution takes place within the Xfilter execution model, where opcode nodes, which are nodes within an XML documents, and plurality of queries are Xpath queries.)(Page 56, section 4.1).

As per claim 17, both Campailla and Altinel teach a query evaluation system, further configured to receive a SOAP (Simple Object Access Protocol) message as the input that is evaluated against the plurality of queries (Page 56, sections 4, 4.1).

As per claim 18, both Campailla and Altinel teach an opcode tree data structure, further comprising a branch node that includes references to more than two dependent opcode nodes (i.e. Figure 3a, titled Example Queries and Corresponding Path Nodes clearly indicate that a branch node is '//b' and more than two dependent opcode nodes are /*/c/d)(Page 57, Figure 3).

As per claim 19, both Campailla and Altinel teach, the executing step further comprising executing a branch node to execute multiple opcode nodes that depend from the branch node, the branch node including an indexed branch lookup function (i.e. Figure 5 depicts an example of executing multiple nodes in Figure 5, section a, titled Example Queries and Corresponding Path Nodes, where the path nodes are multiple opcode nodes that depend from the branch node.)(Page 59, Figure 5).

As per claim 20, both Campailla and Altinel teach a query evaluation system, further comprising an interim results cache that stores results of opcode node executions that are used in the execution of subsequent opcode nodes (Page 56, sections 4, 4.1).

As per claim 21, both Campailla and Altinel teach a query evaluation system, further comprising a filter table that stores the plurality of queries, the filter table further including a reference to the opcode tree (Page 54, section 2.1; page 55, section 2.2; page 56, section 4.1).

As per claim 22, both Campailla and Altinel teach a query evaluation system, an opcode object common to multiple queries further comprising an opcode object that is in a similar location of an opcode object sequence at the beginning of the multiple queries (Page 56, section 4.1).

As per claim 23, both Campailla and Altinel teach, the multiple queries further comprising Xpath queries (i.e. *"The Query Index is used to match documents to individual Xpath queries.")*(Page 56, section 4, 4.1).

As per claim 24, Campailla teaches one or more computer-readable storage media containing computer-executable instructions that, when executed by a computer, perform the following steps (See Figure 2): receiving an input (i.e. input message queue)(See Figure 3; column 5, lines 50-64), wherein the input comprises a plurality of characters (i.e. "*The input message queue model receives the sequence of information messages from a publisher message system…the sequence of information messages include stock quote data, such as a stock quote symbol, etc…."*)(See Figure 3; column 5, lines 50-64); grouping the plurality of characters into one or more elemental language units (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."* The preceding text clearly indicates that the logical query evaluation process includes the step of grouping the plurality of characters into one or more elemental language units.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); breaking the one or more elemental language units into one or more constituent parts (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription*

*module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."*The preceding text clearly indicates that the logical query evaluation process indicates breaking one or more elemental language units into one or more constituent parts.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); generating opcodes from the one or more elemental language units and from the one or more constituent parts (i.e. *"Each of the inverse query subscription modules applies the subscriber's information request criteria to the information..."*)(see Figures 3 and 4; column 6, lines 13-16), wherein the language units have been parsed and compiled into opcodes (Figure 10 illustrates the parsed language units and compiled into opcode nodes.)(See Figures 3, 4, 8, and 10); merging the opcodes into an opcode tree comprising opcode nodes and branch nodes, wherein there are no opcodes added to the opcode tree during an active merging (i.e. Figure 5 appears to create an opcode tree, which is further illustrated in Figure 6, where opcode nodes and branches are illustrated as P1, P2, P3)(See at least Figures 3, 5, 6, 9) traversing an opcode tree (i.e. inverse query decision that is contained within a binary decision diagram)(see Figures 1, 2, and 3; see also column 2, lines 33-45) that includes a plurality of opcode nodes which together define opcodes that should be executed to evaluate a plurality of queries (i.e. *"binary decision diagrams" "evaluates a first information request criteria based upon information within the received messages, evaluates one or more information request criteria based upon information within the received messages using the identified logical implications between one or more binary decision diagrams"* The preceding text clearly indicates that evaluating the information request criteria is to traverse an opcode three that includes a plurality of opcodes.)(column 2, lines 33-45); and wherein a tree segment in a shared path represents an opcode block prefix that is common to two or more queries (i.e. Figure 2 appears to illustrate a tree segment, i.e. P1, P2 in a shared path represents an opcode block prefix, i.e. <rop>, that is common to two or more queries)(Figures 2, 3, 6, and 7).

Campailla does not explicitly teach executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled.

Altinel teaches executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input (i.e. *"The Query Index is used to match documents to individual XPath **queries**." "The events that drive the execution of the Filter Engine are generated by the XML Parser (as described in the following section). In the XFilter **execution** model, a profile is considered to match a document when the final state of its FSM is reached. The Query Index is built over the states of the XPath queries."* The preceding text clearly indicate that execution takes place within the Xfilter execution model, where opcode nodes, which are nodes within an XML documents, and plurality of queries are Xpath queries.)(Page 56, section 4.1); and wherein a relationship between the opcodes (nodes in an XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) and the opcode tree (i.e. XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) is embedded in the opcodes that is created when a query is compiled (see XFilter engine which uses a sophisticated index structure and a modified Finite State Machine approach and represent queries using XPath language)(page 53).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Campailla with the teachings of Altinel to include executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and

wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

The combination of Campailla and Altinel do not explicitly teach the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing.

Sailaja teaches the method wherein the input comprises a plurality of characters (i.e. Figure 1 illustrates the elemental language units, where Parts (Q), (P,R), are elemental language units.)(Figure 1; Sections 1, 3, and 4); evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time (i.e. *"A more clever approach is to devise algorithm that makes a* ***constant number of passes*** *over the document and determine the queries answered by each of its element…"* The Examiner the elements are common query expressions, evaluating the input against multiple queries as determine the queries. The Examiner further understands the limitation of parallel as a constant number of passes, because a naïve way to process queries one at a time is inefficient.)(Section 1, see Example 11); generating at least some of the elemental language units into opcodes (i.e. Figure 1 further exemplifies the generating at least some element units into opcode, where Part (Q), which exemplifies elemental language unit and opcodes which is exemplified by 'Name', 'Brand'. Since Sailaja focuses on XML, an

ordinary person skilled in the art understands that XML tags are sets of instructions that are used to create 'Name' and 'Brand,' but not necessarily limited to just creating 'Name' and 'Brand'.)(Figure 1, Sections 1 and 4); hierarchical nature (i.e. Figure 1 clearly illustrates the hierarchical nature, which is a data tree node showing query labeling. To avoid cluttering of Figure 1, node numbers were omitted, however further explained in Example 11.)(Figure 1; Section 1); maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text and Figures 3 and 4 illiustrates maintaining the opcode tree, which is tracking queries that have been answered and making a copy of the opcode tree is the three different lists QL, CML, and PL which are copies of the data tree node.)(Figure 3; Section 4); and updating the opcode tree copy (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text clearly indicates the use of the auxillary list, PL, which pushes the list. An ordinary person skilled in the art understands that when a push is made, it clearly states that an update is being performed.)(Figure 3; Section 4).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the combined teachings of Campailla and Altinel with the teachings of Sailaja to include the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at

least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing; and updating the opcode tree copy with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

As per claim 25, both Campailla and Altinel teach a computer-readable media, wherein the first segment of opcode nodes includes ancestor opcode nodes of the second opcode node (Page 56, section 4.1; page 57, section 4.2; page 58, sections 4.3, 5).

As perclaim 26, both Campailla and Altinel teach a computer-readable media, the executing opcode nodes further comprising executing each opcode node a single time (i.e. *"The Query Index is used to match documents to individual XPath **queries**." "The events that drive the execution of the Filter Engine are generated by the XML Parser (as described in the following section). In the XFilter **execution** model, a profile is considered to match a document when the final state of its FSM is reached. The Query Index is built over the states of the XPath queries."* The preceding text clearly indicate that execution takes place within the Xfilter execution model, where opcode nodes, which are nodes within an XML documents, and plurality of queries are Xpath queries.)(Page 56, section 4.1).

As per claim 27, both Campailla and Altinel teach, the multiple queries further comprising Xpath queries (i.e. *"The Query Index is used to match documents to individual Xpath queries.")*(Page 56, section 4, 4.1).

As per claim 28, both Campailla and Altinel teach a computer-readable media, further comprising executing one or more branch nodes to execute one or more opcode nodes that depend from the branch node (Page 56, sections 4, 4.1).

As per claim 29, both Campailla and Altinel teach, the executing step further comprising executing a branch node to execute multiple opcode nodes that depend from the branch node, the branch node including an indexed branch lookup function (i.e. Figure 5 depicts an example of executing multiple nodes in Figure 5, section a, titled Example Queries and Corresponding Path Nodes, where the path nodes are multiple opcode nodes that depend from the branch node.)(Page 59, Figure 5).

As per claim 30, both Campailla and Altinel teach a computer-readable media, further comprising executing one or more indexed branch nodes to execute a plurality of opcode nodes that depend from the branch node, the plurality of opcode nodes including a similar comparison function (Page 56, section 4.1).

As per claim 31, both Campailla and Altinel teach, further comprising a hash function as the indexed branch lookup procedure (i.e. Figure 5 depicts an example of a hash table. An ordinary person skilled in the art anticipates the use of a hash function when using a hash table.)(Page 59, Figure 5).

As per claim 32, both Campailla and Altinel teach, further comprising a hash function as the indexed branch lookup procedure (i.e. Figure 5 depicts an example of a hash table. An ordinary person skilled in the art anticipates the use of a hash function when using a hash table.)(Page 59, Figure 5).

As per claim 33, both Campailla and Altinel teach a computer-readable media, further comprising receiving an input that is evaluated against the plurality of queries using the opcode tree (i.e. *"There are two main sets of inputs to the system: user profiles and data items (i.e., documents). User profiles describe the information preferences of individual users. In most systems these profiles are created by the users, typically by clicking on items in a Graphical User Interface."* The preceding text clearly indicates that receiving an input is an input to the system by the user.)(Page 54, section 2.1).

As per claim 34, both Campailla and Altinel teach a computer-readable media, further comprising a compiler configured to execute each query in the plurality of queries to derive the opcode nodes (i.e. *"The Query Index is used to match documents to individual XPath* ***queries****."* *"The events that drive the execution of the Filter Engine are generated by the XML Parser (as described in the following section). In the XFilter* ***execution*** *model, a profile is considered to match a document when the final state of its FSM is reached. The Query Index is built over the states of the XPath queries."* The preceding text clearly indicate that execution takes place within the Xfilter execution model, where opcode nodes, which are nodes within an XML documents, and plurality of queries are Xpath queries.)(Page 56, section 4.1).

As per claim 35, Campailla teaches a method comprising: receiving an input (i.e. input message queue)(See Figure 3; column 5, lines 50-64), wherein the input comprises a plurality of characters (i.e. "*The input message queue model receives the sequence of information messages from a publisher message system…the sequence of information messages include stock quote data, such as a stock quote symbol, etc….*")(See Figure 3; column 5, lines 50-64); grouping the plurality of characters into one or more elemental language units (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."* The preceding text clearly indicates that the logical query evaluation process includes the step of grouping the plurality of characters into one or more elemental language units.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); breaking the one or more elemental language units into one or more constituent parts (i.e. *"Similar logical query evaluation processing is performed for each of the inverse query subscription module" "…subscription process is to use sub-expression implications to leverage commonality between subscriptions..."*The preceding text clearly indicates that the logical query evaluation process indicates breaking one or more elemental language units into one or more constituent parts.)(See Figures 3 and 4; column 5, lines 50-64; column 6, lines 38-55); generating opcodes from the one or more elemental language units and from the one or more constituent parts (i.e. *"Each of the inverse query subscription modules applies the subscriber's information request criteria to the information..."*)(see Figures 3 and 4; column 6, lines 13-16), wherein the language units have been parsed and compiled into opcodes (Figure 10 illustrates the parsed language units and compiled into opcode nodes.)(See Figures 3, 4, 8, and 10); merging the opcodes into an opcode tree comprising opcode nodes and branch nodes, wherein there are no opcodes added to the opcode tree during an active merging (i.e. Figure 5 appears to create an opcode tree, which is further illustrated in Figure 6, where opcode nodes and branches are illustrated as P1, P2,

P3)(See at least Figures 3, 5, 6, 9) traversing an opcode tree (i.e. inverse query decision that is contained within a binary decision diagram)(see Figures 1, 2, and 3; see also column 2, lines 33-45) that includes a plurality of opcode nodes which together define opcodes that should be executed to evaluate a plurality of queries (i.e. *"binary decision diagrams" "evaluates a first information request criteria based upon information within the received messages, evaluates one or more information request criteria based upon information within the received messages using the identified logical implications between one or more binary decision diagrams"* The preceding text clearly indicates that evaluating the information request criteria is to traverse an opcode three that includes a plurality of opcodes.)(column 2, lines 33-45); and wherein a tree segment in a shared path represents an opcode block prefix that is common to two or more queries (i.e. Figure 2 appears to illustrate a tree segment, i.e. P1, P2 in a shared path represents an opcode block prefix, i.e. <rop>, that is common to two or more queries)(Figures 2, 3, 6, and 7).

Campailla does not explicitly teach executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled.

Altinel teaches executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input (i.e. *"The Query Index is used to match documents to individual XPath **queries**." "The events that drive the execution of the Filter Engine are generated by the XML Parser (as described in the following section). In the XFilter **execution** model, a profile is considered to match a document when the final state of its FSM is reached. The Query Index is built over the states of the XPath queries."* The preceding text clearly indicate that execution takes place within the Xfilter execution model, where opcode

nodes, which are nodes within an XML documents, and plurality of queries are Xpath queries.)(Page 56, section 4.1); and wherein a relationship between the opcodes (nodes in an XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) and the opcode tree (i.e. XML document) (Page 54, section 2.2; page 55, section 3; Page 59, section 5.1) is embedded in the opcodes that is created when a query is compiled (see XFilter engine which uses a sophisticated index structure and a modified Finite State Machine approach and represent queries using XPath language)(page 53).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the teachings of Campailla with the teachings of Altinel to include executing each of the opcode nodes in the opcode tree as each opcode node is encountered in the traversal to evaluate the plurality of queries against the input; and wherein a relationship between the opcodes and the opcode tree is embedded in the opcodes that is created when a query is compiled with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

The combination of Campailla and Altinel do not explicitly teach the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing.

Sailaja teaches the method wherein the input comprises a plurality of characters (i.e. Figure 1 illustrates the elemental language units, where Parts (Q), (P,R), are elemental language units.)(Figure 1; Sections 1, 3, and 4); evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time (i.e. *"A more clever approach is to devise algorithm that makes a* ***constant number of passes*** *over the document and determine the queries answered by each of its element…"* The Examiner the elements are common query expressions, evaluating the input against multiple queries as determine the queries. The Examiner further understands the limitation of parallel as a constant number of passes, because a naïve way to process queries one at a time is inefficient.)(Section 1, see Example 11); generating at least some of the elemental language units into opcodes (i.e. Figure 1 further exemplifies the generating at least some element units into opcode, where Part (Q), which exemplifies elemental language unit and opcodes which is exemplified by 'Name', 'Brand'. Since Sailaja focuses on XML, an ordinary person skilled in the art understands that XML tags are sets of instructions that are used to create 'Name' and 'Brand,' but not necessarily limited to just creating 'Name' and 'Brand'.)(Figure 1, Sections 1 and 4); hierarchical nature (i.e. Figure 1 clearly illustrates the hierarchical nature, which is a data tree node showing query labeling. To avoid cluttering of Figure 1, node numbers were omitted, however further explained in Example 11.)(Figure 1; Section 1); maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text and Figures 3 and 4 illiustrates maintaining the opcode tree, which is tracking queries that have been answered and making a copy of the opcode tree is the three different lists QL, CML, and PL which are copies of the data tree node.)(Figure 3; Section 4);

and updating the opcode tree copy (i.e. *"With each data tree node, we associate three lists: the QL (for query labeling) list, which will eventually contain those queries answered by the (subtree rooted at the) node, a list called CML (for chain matching list) that tracks which queries have so far been matched and how far, and an auxillary list called PL (for push list) that is necessary to manage CML…"* The preceding text clearly indicates the use of the auxillary list, PL, which pushes the list. An ordinary person skilled in the art understands that when a push is made, it clearly states that an update is being performed.)(Figure 3; Section 4).

It would have been obvious to a person of ordinary skill in the art at the time of Applicant's invention to modify the combined teachings of Campailla and Altinel with the teachings of Sailaja to include the method wherein the input comprises elemental language units; evaluating the input against multiple queries by evaluating common query expressions of the multiple queries in parallel, at the same time; generating at least some of the elemental language units into opcodes; hierarchical nature; maintaining an opcode tree that is used during query processing by the opcode tree, wherein operations may be undertaken on the opcode tree without interfering with a query processing; and updating the opcode tree copy with the motivation to perform efficient filtering of XML documents for large-scale information dissemination systems (Altinel, Abstract).

As per claim 36, both Campailla and Altinel teach an opcode tree data structure, further comprising a branch node that includes references to more than two dependent opcode nodes (i.e. Figure 3a, titled Example Queries and Corresponding Path Nodes clearly indicate that a branch node is '//b' and more than two dependent opcode nodes are /*/c/d)(Page 57, Figure 3).

As per claim 37, both Campailla and Altinel teach, the executing step further comprising executing a branch node to execute multiple opcode nodes that depend from the branch node, the branch node including an indexed branch lookup function (i.e. Figure 5 depicts an example of executing multiple nodes in Figure 5, section a, titled Example Queries and Corresponding Path Nodes, where the path nodes are multiple opcode nodes that depend from the branch node.)(Page 59, Figure 5).

As per claims 38, both Campailla and Altinel teach, further comprising a hash function as the indexed branch lookup procedure (i.e. Figure 5 depicts an example of a hash table. An ordinary person skilled in the art anticipates the use of a hash function when using a hash table.)(Page 59, Figure 5)

Contact Information

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Farhan M. Syed whose telephone number is 571-272-7191. The examiner can normally be reached on 8:30AM-5:00 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Christian Chace can be reached on 571-272-4190. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Farhan M Syed/
Examiner, Art Unit 2165